Übung 4 - Rekursion

Programmiervorkurs 2025

1 Pfannkuchenrezept

In dieser Aufgabe soll ein Rechner für Pfannkuchenzutaten geschrieben werden, der anhand einer vorhandenen Zutat bestimmt, wie viel man von den restlichen Zutaten benötigt.

Die Zutaten für 2 Personen sind:

- 350g Mehl
- 500ml Milch
- 4 Eier
- 4 TL Zucker
- 4 EL Öl
- 1 TL Backpulver
- 1 Prise Salz

Schreibe ein Programm, dem man die Zutat und die Menge dieser als Eingabe gibt und welches dann die Menge aller anderen Zutaten ausgibt.

Um etwas Tipparbeit zu ersparen, kann der folgende Code-Schnipsel kopiert werden:

```
zutaten = {
    'Mehl': (350, 'g'),
    'Milch': (500, 'ml'),
    'Eier': (4, 'Stück'),
    'Zucker': (4, 'TL'),
    'Öl': (4, 'EL'),
    'Backpulver': (1, 'TL'),
    'Salz': (1, 'Prise')
}

# TODO Hier Code einfügen

# Obige Listen werden (nach Modifikation) ausgegeben
print(f'Für die angegebene Menge werden folgende Zutaten benötigt:')
for name in zutaten:
    menge, einheit = zutaten[name]
    print(f'- {name}: {menge:.1f} {einheit}')
```

2 Rekursion

2.1 Fakultät

1. Schreibe ein Programm, das den Wert des Ausdrucks $1 * 2 * 3 * \ldots * 15 = 15!$ (Fakultät von 15) ohne Nutzung von Schleifen berechnet und das Ergebnis auf der Konsole ausgibt.
2. Erweitere dein Programm so, dass es von beliebigen Eingaben in der Konsole die Fakultät berechnet.

2.2 Fibonacci

Die Fibonacci-Folge ist eine Folge von Zahlen die nach einer festen Regel berechnet werden. Die ersten

beiden Fibonacci-Zahlen sind definiert mit 0 und 1. Jede weitere Fibonacci-Zahl ist die Summe der beiden "'Vorgänger"'-Fibonacci-Zahlen. Der Anfang der Folge ist also 0,1,1,2,3,5,8,13,...

Schreibe eine Funktion, die jeweils die n-te Fibonacci-Zahl mittels Rekursion berechnet. Gib anschließend die ersten 20 Fibonacci-Zahlen aus.

2.3 Gerade oder Ungerade

Schreibe eine der beiden Funktionen *even* und *odd* mithilfe von Rekursion, welche eine Zahl als Parameter entgegennehmen und einen bool zurückgeben, wenn bei *even* die Zahl gerade und bei *odd* die Zahl ungerade ist. Für diese Funktionen dürft ihr weder den Modulo-Operator (%) noch den Divisions-Operator (/) nutzen. Ebenso soll dies mit einer einzigen Funktion umgesetzt werden.

2.4 Pascalsches Dreieck

In dieser Aufgabe soll ein beliebiges Pascalsches Dreieck[1] erzeugt werden.

2.4.1 Berechnung eines Eintrages

Schreibe dafür zunächst eine Funktion, die eine Zahl in diesem Dreieck berechnet. Diese Funktion bekommt zwei Parameter, wobei der erste die Zeile und der zweite die Position in der Zeile repräsentiert. Zur Berechnung soll die Summe zweier Zellen in der darüberliegenden Zeile gebildet werden (eine Animation ist auf der Wikipedia-Seite zu finden). Zeile 0 soll dabei die oberste Zeile sein. Für die n-te Zeile kann der zweite Parameter die Werte 0 bis n annehmen.

2.4.2 Generierung des gesamten Dreiecks

Nun soll ein gesamtes Dreieck generiert werden, dessen Tiefe als Eingabe über die Konsole angegeben werden soll.

Hinweis: Das Padding kann mithilfe von "String-Multiplikation" (`'a' * 4 => 'aaaa'`) umgesetzt werden. Hinweis: Um den üblichen Zeilenumbruch von `print()` zu vermeiden, kann zusätzlich das `end`-Argument gesetzt werden: `print(s, end='')`

2.5 Vergleich Rekursion/Iteration

Bereits in der ersten Aufgabe dieses Übungsblattes und in vorigen Übungsblättern wurde ein Vergleich anhand der Berechnung der Fakultät gezogen. In dieser Aufgabe soll dies nochmal weiter geübt werden. Implementiere dafür das Umdrehen einer beliebigen Liste einmal als Schleife und einmal mithilfe von Rekursion. Das Nutzen von Slices (z.B. `lst[::-1]`) oder die `reverse`-Funktion sind hier nicht erlaubt.

3 ggT und RSA

Zuletzt soll schrittweise die Funktionsweise des RSA-Algorithmus' erklärt veranschaulicht werden. Das Bearbeiten dieser Aufgabe ist optional.

3.1 Größter gemeinsamer Teiler

Zuerst soll eine Funktion definiert werden, mit der man den größten gemeinsamen Teiler (ggT) einer Funk-

[1] https://de.wikipedia.org/wiki/Pascalsches_Dreieck

tion finden kann. Dieser ist wie folgt definiert:

$$ggT(a,b) = \begin{cases} b & a = 0 \\ a & b = 0 \\ ggT(b, a \bmod b) & sonst \end{cases}$$

Teste dein Programm mit verschiedenen Eingaben.

3.2 RSA (Bonus)

Im Folgenden soll nochmal die Nutzung von RSA veranschaulicht werden. Hierfür soll zuerst der Schlüssel berechnet werden und danach die Verschlüsselung und Entschlüsselung implementiert werden.

3.2.1 Schlüsselberechnung

Schreibe eine Funktion `generate_key`, welche zwei Primzahlen p und q entgegennimmt und den öffentlichen und privaten Schlüssel berechnet. Es soll zuerst $\phi(N) = (p-1)(q-1)$ bestimmt werden. Danach soll ein e mit $1 < e < N$ gefunden werden, für das $ggT(e, N) = 1$ gilt. Finde anschließend ein d mit $0 < d < N$ und $e \cdot d \; mod \; N = 1$. Die Funktion soll abschließend ein Tuplepaar zurückgeben: `return ((e, N), (d, N))` (öffentlicher Schlüssel, privater Schlüssel).

3.2.2 Ver- und Entschlüsselung

Für die Verschlüsselung sollen zwei Funktionen geschrieben werden. Die mathematischen Zusammenhänge können dem ersten Übungsblatt entnommen werden.

1. Funktion `encrypt`, welche die Nachricht und den öffentlichen Schlüssel als Argument nimmt und die verschlüsselte Nachricht zurückgibt.
2. Funktion `decrypt`, welche die verschlüsselte Nachricht und den privaten Schlüssel als Argument nimmt und die entschlüsselte Nachricht zurückgibt.